อนันต์ กิตติบรรจุ

วินโดวส์โปรแกรมมิง

ตอนที่ 2

ศึกษาการสร้างแอพพลิเคชันบนวินโดวส์ด้วยโปรแกรม SDK ของไมโครซอฟต์

สวัสดีครับคุณผู้อ่านด้วย คอลัมน์วินโดวส์โปรแกรมมิงนี้จะมีเพื่อน ๆ หลายคนที่จะมาผลัดกันเขียนและแนะนำทีเด็ดหรือเทคนิคต่าง ๆ ที่แต่ละคนเคยเขียนกันมา ส่วนในฉบับนี้เรามาคุยกันถึงเรื่องการสร้างแอพพลิเคชันตัวแรกบนวินโดวส์กันนะครับ

จากคอลัมน์ร่ายยาว GUIs ของคุณพัลลภ คงได้ทำให้ผู้อ่านได้รับทราบความเป็นมาของ GDI (Graphic Device Interface), เครื่องมือที่ใช้ในการพัฒนาโปรแกรม ขั้นตอนการพัฒนาโปรแกรม และโครงสร้างโดยคร่าว ๆ ของแอพพลิเคชันโปรแกรมของวินโดวส์ (จากโปรแกรม SKELETON.C)

ในฉบับนี้ก็คงจะได้ฤกษ์เปิดตัวโปรแกรมตัวแรกกันจริง ๆ เสียทีนะครับ หลังจากที่ผู้อ่านได้ทราบหลักการต่าง ๆ พอสังเขปแล้ว และเพื่อไม่ให้เป็นการเสียเวลาเรามาเริ่มกันเลยนะครับ

โครงสร้างหลักของ HELLOWIN.C

Include files
Function prototypes
Variables
Main function WinMain( )
Initialization function InitApplication( )
Creating Window InitInstance( )
Window function MainWndProc( )
Paint function PaintFunc( )

Include files

```
# include <windows.h>
# include "hellowin.h"
```

สำหรับไฟล์ WINDOWS.H นั้น เป็นไฟล์ที่มาพร้อมกับชุด SDK ของไมโครซอฟต์และจะต้องใส่เข้าไว้ในซอร์สโค้ดเราเสมอ ไฟล์นี้มีขนาดค่อนข้างใหญ่มาก (108, 580 ไบต์) ซึ่งประกอบไปด้วยค่าคงที่, ชนิดข้อมูลแบบโครงสร้าง (Structure type), แมคโคร, ชนิดของแมสเสจต่าง ๆ และโปรโตไทพ์ฟังก์ชันของวินโดวส์ทั้งหลายเป็นต้น

ส่วนไฟล์ HELLOWIN.H เป็นส่วนโปรโตไทพ์ฟังก์ชันของ HELLOWIN.C

Variables

เป็นส่วนตัวแปรโกลบอลของโปรแกรม ซึ่งมีดังรูปที่ 1

Function Prototype

เป็นส่วนที่ประกาศเอาไว้ว่าฟังก์ชันทั้งหลายใน HELLOWIN.C มีอะไรบ้าง

```
HANDLE  hInst;                                         // Instance of application
HWND    hWndMain;                                      // Main window
char    szApplName[  ]=“Hellowin”;                     // Application name
char    szTitleName[  ]=“Sample First Application”;    // Title name
char    szMessage[  ]=“Hello, Windows”;                // Message to displayed
```

รูปที่ 1 ตัวแปรโกบอลในโปรแกรม

รีเทอร์นชนิดตัวแปรแบบใด และมีพารามิเตอร์และชนิดใดอยู่บ้าง การทำเช่นนี้มีประโยชน์มาก เพราะเป็นการลดข้อผิดพลาดของโปรแกรมและการรายงานข้อผิดพลาด (warning) ขณะที่ทำการคอมพายล์ และด้านล่างนี้เป็นส่วนโปรโตไทพ์ HELLOWIN.H ครับ

```
int PASCAL WinMain(HANDLE,HANDLE,
HANDLE, LPSTR, int);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
void PaintFunc(HWND);
long FAR PASCAL MainWndProc(HWND,
unsingned, WORD, LONG);
```

Main function

ส่วนนี้คือฟังก์ชัน WinMain( ) ฟังก์ชันนี้ถ้ามองกันคร่าว ๆ แล้วก็คือส่วน main( ) ในโปรแกรมภาษาซีทั่ว ๆ ไปนั่นเอง กล่าวคือ เป็นจุดเริ่มต้นของการทำงานที่ตัววินโดวส์จะเรียกเข้ามา เมื่อเริ่มรันแอพพลิเคชันของเราซึ่งมีลักษณะการทำงานดังรูปที่ 2

พารามิเตอร์ hPrevInstance จะเป็นตัวบอกว่า แอพพลิเคชันนี้ถูกเปิดมาแล้วหรือยัง ถ้ายังค่านี้จะเป็นศูนย์ต้องทำการเรียกฟังก์ชัน InitApplication ( ) เสียก่อน

Registering window class

เป็นการลงทะเบียน (registering) ให้กับแอพพลิเคชัน เพื่อให้วินโดวส์รู้จักคลาสนี้ โดยก่อนที่จะสร้างหน้าต่างใด ๆ ขึ้นมาจะต้องบอกไปด้วยว่าจะใช้คลาสอะไรซึ่งคลาสนี้จะมีส่วนหนึ่งที่วินโดวส์เองก็ใช้และได้ลงทะเบียนไว้ก่อนแล้ว ที่เรียกกันว่า Predefined window class อันได้แก่ EDIT, BUTTON, LISTBOX, SCROLLBAR และ STATIC เป็นต้น หากเราสร้างหน้าต่างที่มีคลาสตัวที่วินโดวส์รู้จักแล้ว (จาก predefined window class หรือจากการที่เราทำ Registering window class ไปแล้ว) เราสามารถที่จะสร้างวินโดว์คลาสเหล่านี้ได้ทันทีโดยไม่ต้องมาลงทะเบียนซ้ำอีก

วิธีการลงทะเบียนเพื่อสร้างคลาสใหม่จะเรียกผ่านฟังก์ชัน

```
BOLL RegisterClass (LPWND CLASS
lpWndClass)
```

จากโปรแกรมมิงบนวินโดวส์ฉบับที่แล้ว ผู้อ่านคงจำเรื่อง Hungarian Notation กันได้ นั่นคือ ipWndClass หมายถึง Long pointer to WNDCLASS โครงสร้างชนิดข้อมูลของ WNDCLASS เป็นรูปที่ 3

ซึ่งมีฟิลด์ที่น่าพูดถึง ได้แก่

lpfnWndProc จะชี้ไปยัง Window function ที่เป็นเจ้าของคลาสนี้ (ในที่นี้คือ MainWndProc ( ))

hInstance เป็นค่าอินสแตนท์ของแอพพลิเคชัน

hIcon เป็นค่าแฮนเดิลของไอคอนที่จะใช้เป็นสัญลักษณ์ของคลาส (เราสามารถใช้ชุดเครื่องมือของ SDK คือ

```
int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE hInstance;              // current instance
HANDLE hPrevInstance;          // previous instance
LPSTR lpCmdLine;               // command line
int nCmdShow;                  // show-window type (open/icon)
{
    MsG msg;                                // message
    if (!hPrevInstance)                     // other instances of app running?
    if (!InitApplication(hInstance))        // Initialize shared things
      return (FALSE);                       // Exits if unable to initialize
    /* Perform initializations that apply to a specific instance */
    if (!InitInstance(hInstance, nCmdShow))
      return (FALSE);
    /* Acquire and dispatch messages until a WM _ QUIT message is receive. */
    while (GetMessage(&msg,    // message structure
        NULL,                  // handle of window receiving the message
        NULL,                  // lowest message to examine
        NULL))                 // highest message to examine
    {
    TranslateMessage(&msg);    // Translates virtual key codes
    DispatchMessage(&msg);     // Dispatches message to window
    }
    return (msg.wParam);       // Returns the value from PostQuitMessage
```

รูปที่ 2 ฟังก์ชัน WinMain ( )

```
Typedef struct tag WNDCLASS
{
    WORD style;                            // window class type
    Long (FAR PASCAL *IpfnWndProc) ( )     // pointer to window function
    int cbClsErtra;                        // extra bytes - end of structure
    int cbWndExtra;                        // extra bytes - end of window instance
    HANDLE hInstance;                      // window class instance
    HICON hIcon;                           // window class icon
    HCURSOR hCursor;                       // window class cursor
    HBRUSH hBrBackground;                  // background color
    LPSTR ipszMenuName;                    // menu associated with window class
    LPSTR LpszClassName                    // window class name
}WNDCLASS;
```

รูปที่ 3 โครงสร้างชนิดข้อมูลของ WNDCLASS

SDKPAINT หรือทูลส์ตัวใด ๆ ก็ได้มาสร้าง) ซึ่งตอนที่แอพพลิเคชันถูกทำให้เล็กสุด (minimize) วินโดวส์จะนำรูปไอคอนนี้มาโชว์แทน แต่ตอนนี้เราจะใช้ไอคอนจาก GDI ของวินโดวส์ไปก่อนคือ เรียก

```
LoadIcon (NULL, IDI.APPLICATION)
```

hCursor เป็นค่าแฮนเดิลของเคอร์เซอร์ และเช่นกันเราสามารถสร้างโดยใช้ชุดเครื่องมือของ SDK มาสร้าง หรือใช้ของ GDI ก็ได้ ซึ่งมีอยู่หลายตัว เช่น รูปลูกศร (ARROW), รูปนาฬิกาทราย (Hour glass) ในแอพพลิเคชันนี้จะใช้รูปลูกศร ซึ่งถือเป็นเคอร์เซอร์ที่ใช้กันทั่วไป โดยเรียก

```
Load Cursor (NULL, IDC_ARROW)
```

hBrBackground เป็นแฮนเดิลของแปรง (Brush) ที่จะให้ GDI ใช้ทาลงไปบนพื้นวินโดว์ ในที่นี้เราจะใช้แปรงของ GDI ไปก่อนและเป็นแปรงสีขาว โดยเรียก

```
GetStockObject (WHITE_BRUSH)
```

lpszMenuName เป็น long pointer to string ซึ่งเป็นชื่อของเมนูที่ปิดท้ายด้วยไบต์ศูนย์ ("\ o") ต้องเป็นชื่อที่อยู่ในส่วน resource file ส่วนนี้ไว้จะอธิบายในเรื่องเมนูต่อไป ส่วนในแอพพลิเคชันนี้ยังไม่มีเมนู ดังนั้นจะเซตค่าเป็น NULL

lpszClassName เป็น long pointer to string อันเป็นชื่อของคลาสที่จะลงทะเบียนนี้

หลังจากเซตค่าทุกตัวเรียบร้อยแล้ว จึงเรียก RegisterClass ( ) ถ้าสำเร็จจะคืนค่าที่ไม่ใช่ศูนย์ แต่ถ้าไม่สำเร็จจะคืนค่าศูนย์กลับไป

Creating Window

หลังจากที่ได้คลาสมาเรียบร้อยแล้ว เราก็พร้อมที่จะสร้างหน้าต่างได้แล้ว รูปแบบการสร้างหน้าต่างเป็นดังรูปที่ 4

ส่วนใน HELLOWIN.C จะส่งพารามิเตอร์ ดังรูปที่ 5

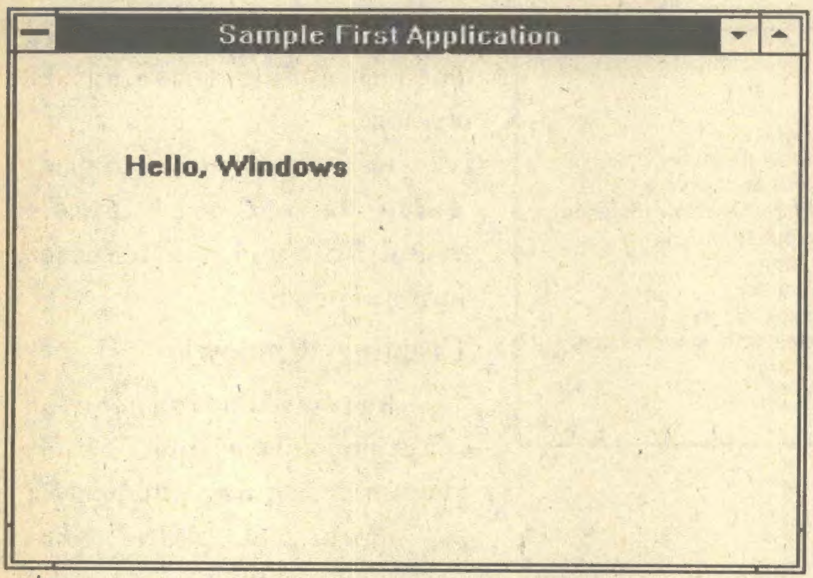
ให้สังเกตฟิลด์ dwStyle ซึ่งได้เซต

```
HWND CreateWindow (lpClassName,   // คลาสที่ได้ลงทะเบียนแล้ว
                  lpWindowName,  // ชื่อของ title bay หรือ caption bar
                  dwStyle,       // เป็น window style
                  X,             // X-Coordinate นับจากมุมซ้ายบน
                  Y,             // Y-Coordinate นับจากมุมซ้ายบน
                  nWidth         // ความกว้างของหน้าต่าง
                  nWndParent,    // ความสูงของวินโดว์
                  nWndParent,    // วินโดว์ที่เป็น parent
                  nMenu,         // ค่าแฮนเดิลของเมนู. ถ้า NULL คือใช้เมนูจากคลาส
                  nInstance,     // window instance
                  lpParam);      // พารามิเตอร์ที่จะส่งให้พร้อมกับ WM.CREAT
```

รูปที่ 4 รูปแบบการสร้างหน้าต่าง

```
hWndMain = CreateWindow(
        szApplName,              /* See RegisterClass( ) call.             */
        szTitleName,             /* Text for window title bar.             */
        WS_OVERLAPPEDWINDOW,     /* Window style.                          */
        CW_USEDEFAULT,           /* Default horizontal position.           */
        CW_USEDEFAULT,           /* Default vertical position.             */
        CW_USEDEFAULT,           /* Default width.                         */
        CW_USEDEFAULT,           /* Default height.                        */
        NULL,                    /* Overlapped windows have no parent.     */
        NULL,                    /* Use the window class menu.             */
        hInstance,               /* This kinstance owns this window.       */
        NULL                     /* Pointer not needed.                    */
);
```

รูปที่ 5 การส่งพารามิเตอร์ใน HELLOWIN.C



Hello, Windows

รูปที่ 6

ให้เป็น WS_OVERLAPPEDWINDOW เป็นสไตล์ที่ใช้กันทั่ว ๆ ไป ประกอบด้วย Minimize, Maximize box, Caption bar และ System Menu (ดังรูปที่ 6) หลังจากเรียก CreatWindow( ) เสร็จก็จะต้องเรียก

```
/* Make the window visible; update its
client area; and return ''success'' */
ShowWindow(hWndMain, nCmdShow);
/* Show the window */
UpdateWindow(hWndMain);
/* Sends WM _ PAINT message */
```

เพื่อให้วินโดวส์แสดงหน้าจอโดย hWndMain เป็นค่าแฮนเดิลวินโดว์ที่ได้รับ เมื่อตอนที่สร้างหน้าต่างและ nCmdShow เป็นพารามิเตอร์ตัวที่ 4 จากฟังก์ชัน WinMain( ) ซึ่งวินโดวส์เรียกเมื่อตอนที่เริ่มรันแอพพลิเคชัน โดยค่านี้จะเป็นตัวบอกให้การแสดงหน้าต่างนี้เป็นหน้าจอวินโดว์เต็มรูปแบบเลย (กรณีที่เราเรียกแอพพลิเคชันจากการใช้เม้าส์กดดับเบิลคลิ๊ก หรือกดคีย์ <Enter> เป็นต้น) หรืออีกแบบจะแสดงเป็นไอคอน (กรณีนี้เกิดจากการกดคีย์ <Shift> + <Enter> หรือแอพพลิเคชันบางตัวอาจจะสั่งให้วินโดวส์ทำการโหลดแอพพลิเคชัน เมื่อเริ่มต้นเข้าสู่วินโดวส์เลยก็ได้ โดยการสร้างไว้ในไฟล์ WIN.INI)

ส่วนการเรียก UpdateWindow( ) ก็เพื่อให้วินโดวส์ส่งคำสั่ง WM_PAINT เพื่อให้ตัวแอพพลิเคชันทำการวาดส่วนที่ต้องการแสดงในไคลเอนต์แอเรีย ซึ่งจะเป็นการแสดงข้อความ ''Hello, Windows'' ดังรูปที่ 6

โดยเมื่อ HELLOWIN ได้รับแมสเสจ WM_PAINT จากวินโดวส์จะเรียกฟังก์ชัน PaintFunc(hWnd) ซึ่งมีการทำงานดังนี้

```
hDC = Begin Paint (hWnd, &ps);
// Get display context of main Window
```

ค่า ps มีชนิดข้อมูลเป็น PAINSTRUCT ซึ่งจะเก็บส่วนสำคัญที่จำเป็นต่อการวาดนี้ เช่น ส่วนในไคลเอ็นต์แอเรียที่จำเป็นต้องได้รับการวาดใหม่ (เพราะการวาดไม่จำเป็นต้องวาดทั้งหมดจะวาดเฉพาะส่วนที่ต้องอัพเดตเท่านั้น), แฟลกที่บอกว่าจะต้องเคลียร์พื้นหน้าต่างก่อนจะวาด เป็นต้น ในขณะนี้ขอให้ผู้อ่านเพียงทราบไว้ก่อนว่า เราเรียก BeginPaint( ) เพื่อจะรับค่าดิสเพลย์คอนเท็กซ์มาใช้วาดก่อน

```
Textout (hDc, 50, 50, (LPSTR) szMessage,
lstrlen (szMessage);
```

ซึ่งจะแสดงข้อความว่า ''Hello, Windows'' บนไคลเอ็นต์แอเรียนับจากมุมซ้ายบนสุดไปทางขวา (x) 50 จุดและลงมาข้างล่าง (y) อีก 50 จุด

```
EndPaint (hWnd, &ps);
```

ซึ่งจะทำให้วินโดวส์รับรู้ว่าส่วนไคลเอ็นต์แอเรียที่ยังไม่ได้รับก่อนอัพเดตนั้นได้อัพเดตเป็นที่เรียบร้อยแล้ว วินโดวส์ก็จะปลดปล่อย (free) ส่วนดิสเพลย์คอนเท็กซ์นี้ต่อไป พึงระลึกไว้เสมอนะครับว่าเมื่อมี BeginPaint ก็ต้องจบด้วย EndPaint นะครับ

Message Loop

เรื่องแมสเสจจัดเป็นอีกส่วนหนึ่งที่ผู้เขียนแอพพลิเคชันบนวินโดวส์จะต้องทำความเข้าใจให้กระจ่างเพราะเป็นหัวใจของวินโดวส์ทีเดียว จะทำให้แต่ละแอพพลิเคชันสามารถที่ทำงานไปพร้อม ๆ กันได้ (multitasking) ในร่ายยาว GUIs ที่คุณพัลลภได้เขียนไว้หลายตอนและได้กล่าวถึงหลักการเรื่องแมสเสจและส่วนแมสเสจลูปไว้ด้วย (ในไมโครฯ 75, 77 และ 78) สำหรับวิธีการเขียนแมสเสจลูปแสดงได้ดังนี้

```
/* Acquire and dispatch messages until a
WM _ QUIT message is received. */
while (GetMessage(&msg, /* message
                         structure */
NULL, /* handle of window receiving the
                              message */
NULL, /* lowest message to examine */
NULL)) /* highest message to examine */
{
TranslateMessage (&msg);  /* Translates
                     virtual key codes */
DispatchMessage (&msg);  /* Dispatches
                     message to window */
}
```

Module Definition file

หลังจากที่ได้อธิบาย HELLOWIN.C คราวนี้จะมาศึกษาส่วน Module definition file คือไฟล์ HELLOWIN.DEF

```
NAME      hellowin
DESCRIPTION 'Sample First Application'
EXETYPE  WINDOWS
STUB   'WINSTUB.EXE'
CODE   PRELOAD MOVEABLE
       DISCARDABLE
DATA   PRELOAD MOVEABLE
       MULTIPLE
HEAPSIZE   4096
STACKSIZE  4096
EXPORTS
   MainWndProc
```

NAME เป็นชื่อของแอพพลิเคชัน ซึ่งควรจะมีชื่อตรงกับชื่อแอพพลิเคชัน

DESCRIPTION เป็นคอมเมนต์อย่างย่อของแอพพลิเคชัน

EXETYPEต้องเป็นวินโดวส์เสมอ เพื่อให้ลิงค์เกอร์รู้ว่าเป็นแอพพลิเคชันของวินโดวส์ที่มีลักษณะแตกต่างจาก EXE ทั่ว ๆ ไปบนดอสที่ว่าส่วน Header และ Entrypoint ไม่เหมือนกัน

STUB จะบอกว่าจะใช้โปรแกรมของดอสตัวไหน (ที่มีนามสกุล .EXE เท่านั้น) มารันแทนเมื่อมีผู้พยายามที่จะเรียกแอพพลิเคชันตัวนี้บนดอส โดยทั่วไปแล้วมักใช้ WINSTUB.EXE ที่จะดิสเพลย์ข้อความ

```
"This program reguires Microsoft Win-
dow."
```

CODE ฟิลด์นี้จะบอกการจัดการในส่วนโค้ดเซกเมนต์ (Code Segment) อย่างไร ซึ่งมีอยู่หลายรูปแบบ ได้แก่

PRELOAD ให้เซกเมนต์ถูกโหลดทันทีเมื่อเข้าสู่แอพพลิเคชันนี้

LOADONCALL ให้เซกเมนต์ถูกโหลดเมื่อถูกเรียกใช้เท่านั้น

MOVEABLE เป็นการยอมให้ตัวจัดการเมมโมรีของวินโดวส์จัดการเคลื่อนย้ายได้เมื่อมีความจำเป็น

FIXED ตรงข้าม MOVEABLE คือจะไม่ยอมให้เซกเมนต์เคลื่อนย้ายไปไหน เมื่อมันถูกโหลดไว้ที่ใดแล้วจะอยู่ที่นั่นไปตลอดจนกว่าจะปิดแอพพลิเคชัน

DISCARDABLE เป็นการยอมให้ตัวจัดการเมมโมรีของวินโดวส์ นำเซกเมนต์ออกไปก่อนได้ เมื่อมันโหลดมาแล้วแต่ยังไม่ถูกเรียกใช้ในตอนนั้น

DATA ฟิลด์นี้จะบอกการจัดการในส่วนดาต้าเซกเมนต์ (Data Segment) ซึ่งมักใช้ PRELOAD และ MOVEABLE เช่นเดียวกับส่วน CODE แต่จะแตกต่างกันที่มี MULTIPLE ซึ่งหมายถึงแต่ละอินสแตนท์ของแอพพลิเคชันจะมีส่วนดาต้าเซกเมนต์ของตนเองไม่ใช้รวมกัน หากจะยอมให้ดาต้าเซกเมนต์ใช้รวมกันได้ต้องใช้ SINGLE และถ้าเป็น NONE หมายถึงไม่มีดาต้าเซกเมนต์เลย (ไม่มีตัวแปรแบบโกลบอล)

HEAPSIZE เป็นขนาดของ local heap ที่น้อยที่สุด มีหน่วยเป็นไบต์ โดยทั่วไปค่านี้มักเซตไว้อย่างต่ำ 4 กิโลไบต์ (4096 ไบต์)

STACKSIZE ขนาดของ local stack เป็นส่วนที่ใช้เก็บพารามิเตอร์ เมื่อมีการเรียกฟังก์ชัน ใช้เก็บตัวแปร และใช้เมื่อมีการเรียกฟังก์ชันแบบรีเคอร์ซีฟ ค่านี้ควรจะเซตไว้อย่างต่ำ 5 กิโลไบต์

EXPORTS เป็นฟิลด์ที่มีความสำคัญอีกตัวหนึ่งที่ผู้เขียนโปรแกรมมักลืมประกาศไว้เสมอ ซึ่งจะทำให้แอพพลิเคชันเกิด UAE (Unrecovable Application Error) ได้ ส่วนที่ต้องประกาศไว้ในฟิลด์นี้ได้แก่ฟังก์ชันที่ถูกเรียกโดยเคอร์เนลของวินโดวส์ (Callback function) หากมีฟังก์ชันเหล่านี้มาก ๆ ควรจะมีเครื่องหมาย “@” แล้วตามด้วยหมายเลข เช่น “MainWinProc @ 1” เป็นต้น เพราะเป็นการประหยัดเนื้อที่มากขึ้น แทนที่วินโดวส์จะเก็บเป็นชื่อหลายไบต์กลับใช้เพียง 2 ไบต์ นั่นเอง

Make file

เป็นไฟล์สุดท้ายแล้วที่เราจะมาทำความเข้าใจกันสักหน่อย ก่อนที่จะได้สร้าง HELLOWIN.EXE ไฟล์นี้คือ HELLOWIN.MAK ซึ่งเป็นไฟล์อินพุตให้กับ NMAKE.EXE (มาจาก new make เป็นโปรแกรมเมคไฟล์บนดอสของไมโครซอฟต์ซีคอมพายเลอร์ เวอร์ชัน 6.0) เพื่อให้มันทำการคอมพายล์และลิงค์

ประโยชน์ของการสร้างเป็นเมคไฟล์นี้ เพราะต้องการลดเวลาที่ต้องใช้ไปในขณะที่คอมพายล์ไม่ให้คอมพายล์บ่อย ๆ คอมพายล์เฉพาะไฟล์ที่มีการเปลี่ยนแปลงเท่านั้น โดยเราจะต้องสร้างกฎ (rule) และการกระทำ (action) ขึ้นมา และนี่เป็นเมคไฟล์ของ HELLOWIN

```
Model = S
Warn  = 3

ALL: hellowin.exe

hellowin.obj: hellowin.h hellowin.c
  cl -c -A$(Model) -W$(Warn) -Gsw -Zp
  -Os hellowin.c

hellowin.exe: hellowin.obj  hellowin.def
  hellowin.res
```

```
link /NOD hellowin,hellowin.exe,NUL,
libw + $(Model)libcew,hellowin.def
rc hellowin.exe
```

Model = S เป็นการเซตให้ขนาดเมมโมรีเป็นแบบ Small

Warn = 3 เป็นระดับของคำเตือน (Warning) ที่จะให้คอมพายเลอร์รายงานมีด้วยกันทั้งหมด 5 ระดับ แต่เมื่อนำมาใช้คอมพายล์ไฟล์ของวินโดวส์ จะเซตได้สูงสุดแค่ 3 เท่านั้น (เกินกว่านี้คอมพายล์ WINDOWS.H ไม่ผ่าน)

ALL : hellowin.exe กฎนี้จะต้องนำมาไว้เป็นกฎแรกเสมอ เพื่อบอกให้ NMAKE ทราบว่า จะดูไฟล์อะไรเป็นตัวแรก ซึ่งควรจะเป็นชื่อของแอพพลิเคชันเสมอ

hellowing.obj เป็น target file จากการคอมพายล์ HELLOWIN.C ตามกฎ “hellowin.obj : hellowin.h hellwin.c” บอกว่าให้เทียบไฟล์นี้กับ HELLOWIN.C และ HELLOWIN.H ถ้า 2 ไฟล์นี้มีวันที่และเวลาที่ใหม่กว่า ก็จะทำแอคชัน “CL -C -A$ (Model) -W$(warn) -GSW -Zp -Os hellowing.c”

ออปชันที่สำคัญ

สำหรับออปชัน -A$(Model) ตัว $ (Model) คือการอ้างตัวแปร Model ในนี้เซตไว้เป็น “S” ดังนั้นจะได้เป็น “-AS” ซึ่ง -AS, -AC, -AM, -AL เป็นขนาดเมมโมรีโมเดลแบบ Small, Compact, Medium และ Large ตามลำดับ ซึ่งผมคงไม่ต้องกล่าว ณ ที่นี้ เพราะเชื่อว่าผู้อ่านคงทราบกันดีแล้ว

– FPi เป็นการเลือกใช้ Coprocessor / Emulator ในการจัดการในส่วนคำนวณ Floating Poing ออปชันนี้จะดีฟอลต์อยู่แล้ว คือถ้าคอมพายเลอร์พบว่ามี Coprocessor (โปรเซสเซอร์ตระกูล 80xx7) มันจะใช้คำสั่งของโปรเซสเซอร์เลย แต่หากไม่มีมันจะอีมูเลตโดยซอฟต์แวร์ ดังนั้นจึงควรใส่ไลบรารี “WIN87.LIB” ไว้ด้วยหากมีการใช้ Floating Point

– C คือให้คอมพายล์เป็นออปเจ็กต์ไฟล์เท่านั้น ไม่ต้องลิงค์

– Gw ให้คอมพายเลอร์สร้างโค้ดพิเศษเพิ่มแบบ prolog และ epilog สำหรับฟังก์ชันแบบไกล (far function) ทั้งหมด เมื่อให้วินโดวส์สามารถทำการเคลื่อนย้ายโค้ดและดาต้าเซกเมนต์ (ออปชันบังคับ)

– Gs ไม่ให้มีการตรวจสแต็กทุก ๆ ครั้งที่มีการเรียกฟังก์ชัน

– Zp ทำให้ข้อมูลในแต่ละฟิลด์ของโครงสร้างข้อมูลแต่ละตัวนั้นลงตัวไบต์ (byte boundary) ออปชันนี้วินโดวส์บังคับให้ใส่ไว้ เพราะต้องใช้กับ HELLOWIN.H ด้วย

– Zi ให้สร้าง symbolic information ใช้เมื่อต้องการดีบัก ใน Microsoft Code View

– Os ให้ทำการออปติไมซ์โค้ดตามขนาด (โดยปกติคอมพายเลอร์จะออปติไมซ์ตามความเร็ว) ซึ่งวินโดวส์แนะนำให้ใช้ตัวนี้

– Od ไม่ให้ทำการออปติไมซ์ (ใช้เมื่อจะดีบักโปรแกรม)

ส่วน LINK.EXE มีรูปแบบการใช้งานดังนี้

LINK <Options> object file(S), // แยกออปเจ็กต์แต่ละตัวด้วย “+” หรือช่องว่าง

<EXE file>, // ชื่อของเอ็กซีคิวต์ไฟล์

<MAP file>, // ชื่อแม็พไฟล์

<Library files>, // แยกแต่ละตัวด้วย “+” หรือช่องว่างเช่นกัน

Module definition file

คำอธิบายส่วนออปชัน

/CO บอกให้ลิงค์เกอร์ใส่ symbolic information ใช้เพื่อทำการดีบัก

/NOD มาจากคำว่า No default Library Search ออปชันนี้วินโดวส์บังคับให้ใส่ไว้เสมอ เพื่อให้ลิงค์เกอร์ใช้ standard C runtine library เพราะมันไม่คอมแพตกับวินโดวส์ และนี่เป็นรูปแบบการลิงค์ของ HELLOWIN.MAK

```
link  /NOD  hellowin,hellowin.exe,NUL,
libw + $(Model)libcew,hellowin.def
rc hellowin.exe
```

สังเกตส่วนแม็พไฟล์ที่เซตไว้เป็น “NUL” ไว้ไม่ให้ลิงค์เกอร์สร้างแม็พไฟล์ และส่วนในฟิลด์ไลบรารี จะมีไลบรารี slibcew.lib อันเป็นไลบรารีสำหรับโมเดลสมอลล์ที่มาพร้อมกับชุด SDK ของไมโครซอฟต์และอีกไลบรารีคือ libw.lib ซึ่งไลบรารีทั้งสองตัวนี้จะต้องใส่ไว้ตลอดนะครับ และที่เห็น “rc hellowin.exe” นั้นเป็นการเรียกตัว resource compiler มาเพิ่มโค้ดบางอย่างเข้าไป เพื่อไม่ให้วินโดวส์รายงานว่าแอพพลิเคชันนี้อาจแครชได้ (โดยปกติเราจะต้องมีไฟล์ resource อยู่ ซึ่งคงจะพูดในครั้งต่อไป)

ส่งท้าย

หลังจากที่ผู้เขียนได้อธิบายโปรแกรมมาอย่างละเอียดเลยทีเดียว ก็คงได้เวลาที่ผู้อ่านจะลองสร้างแอพพลิเคชัน HELLOWIN.EXE กันได้แล้วครับ สำหรับซอร์สโค้ดโปรแกรมทั้งหมด ผู้เขียนได้รวมเข้าไว้ในบทความตอนนี้ด้วย หวังว่าผู้อ่านคงจะได้ทดลองใช้กันด้วยนะครับ จะได้เข้าใจได้ดีขึ้น แล้วจะรู้ว่าการสร้างแอพพลิเคชันบนวินโดวส์นั้นก็ไม่ได้ยากเย็นอะไรเลย แล้วคราวหน้าเราจะมาคุยเรื่อง “PAINTING WINDOW” กันต่อนะครับ

โปรแกรม HELLOWIN.C

```
    Written by...
       Anan Kittibunlu
    Date written...
       Jan/1992
    Purpose ...


  This program demonstrates the first basic programming of Windows applications. It
  displays the words "Hello, Windows" on the window
  ******************************************************************************/

/* --------------------------< Include files >--------------------------- */

#include <windows.h>
#include "hellowin.h"

/* ------------------------< Global variables >-------------------------- */

HANDLE hInst;                                          // Instance of application
HWND   hWndMain;                                       // Main window
char   szApplName[]="Hellowin";                        // Application name
char   szTitleName[]="Sample First Application";       // Title name
char   szMessage[] = "Hello, Windows";                 // Message to be displayed

/******************************************************************************
 W i n M a i n ()
 ================

    Windows recognizes this function by name as the initial entry point
    for the program.  This function calls the application initialization
    routine, if no other instance of the program is running, and always
    calls the instance initialization routine.  It then executes a message
    retrieval and dispatch loop that is the top-level control structure
    for the remainder of execution.  The loop is terminated when a WM_QUIT
    message is received, at which time this function exits the application
    instance by returning the value passed by PostQuitMessage().

    If this function must abort before entering the message loop, it
    returns the conventional value NULL.

******************************************************************************/
int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE hInstance;        // current instance
HANDLE hPrevInstance;    // previous instance
LPSTR lpCmdLine;         // command line
int nCmdShow;            // show-window type (open/icon)
{
    MSG msg;                               // message

    if (!hPrevInstance)                    // Other instances of app running?
    if (!InitApplication(hInstance))       // Initialize shared things
        return (FALSE);                    // Exits if unable to initialize

    /* Perform initializations that apply to a specific instance */

    if (!InitInstance(hInstance, nCmdShow))
        return (FALSE);

    /* Acquire and dispatch messages until a WM_QUIT message is received. */

    while (GetMessage(&msg, // message structure
        NULL,               // handle of window receiving the message
        NULL,               // lowest message to examine
        NULL))              // highest message to examine
    {
    TranslateMessage(&msg); // Translates virtual key codes
    DispatchMessage(&msg);  // Dispatches message to window
    }
    return (msg.wParam);    // Returns the value from PostQuitMessage
}

/******************************************************************************
 I n i t A p p l i c a t i o n ( )
 ===================================

   This function is called at initialization time only if no other
   instances of the application are running.  This function performs
   initialization tasks that can be done once for any number of running
   instances.

   In this case, we initialize a window class by filling out a data
   structure of type WNDCLASS and calling the Windows RegisterClass()
   function.  Since all instances of this application use the same window
   class, we only need to do this when the first instance is initialized.

******************************************************************************/
int InitApplication (HANDLE hInstance)
{
 WNDCLASS MainWndClass;                        // Main window class

 /* ---------------< Specify window class information >------------------ */

 MainWndClass.lpszClassName = szApplName;                     // Window class
 MainWndClass.hInstance     = hInstance;                      // Instance
 MainWndClass.lpfnWndProc   = MainWndProc;                    // Window function
 MainWndClass.style         = NULL;                           // Class style
 MainWndClass.lpszMenuName  = NULL;                           // No menu
 MainWndClass.hCursor       = LoadCursor(NULL, IDC_ARROW);    // Mouse cursor
 MainWndClass.hIcon         = LoadIcon(NULL, IDI_APPLICATION); // Default icon
 MainWndClass.hbrBackground = GetStockObject(WHITE_BRUSH); // White background
 MainWndClass.cbClsExtra    = 0;                              // No extra bytes
 MainWndClass.cbWndExtra    = 0;                              // No extra bytes

 /* ---------------< Registering "Structur" window class >---------------- */

 return (RegisterClass(&MainWndClass));
```

โปรแกรม HELLOWIN.C (ต่อ)

```
/******************************************************************************
 I n i t I n s t a n c e ( )
 ====================================

    This function is called at initialization time for every instance of
    this application.  This function performs initialization tasks that
    cannot be shared by multiple instances.

    In this case, we save the instance handle in a static variable and
    create and display the main program window.

******************************************************************************/
OOL InitInstance(hInstance, nCmdShow)
    HANDLE          hInstance;          // Current instance identifier.
    int             nCmdShow;           // Param for first ShowWindow() call.


    /* Save the instance handle in static variable, which will be used in  */
    /* many subsequence calls from this application to Windows.            */

    hInst = hInstance;

    /* Create a main window for this application instance.  */

    hWndMain = CreateWindow(
        szApplName,                         /* See RegisterClass() call.      */
        szTitleName,                        /* Text for window title bar.     */
        WS_OVERLAPPEDWINDOW,                /* Window style.                  */
        CW_USEDEFAULT,                      /* Default horizontal position.   */
        CW_USEDEFAULT,                      /* Default vertical position.     */
        CW_USEDEFAULT,                      /* Default width.                 */
        CW_USEDEFAULT,                      /* Default height.                */
        NULL,                               /* Overlapped windows have no parent.*/
        NULL,                               /* Use the window class menu.     */
        hInstance,                          /* This instance owns this window.*/
        NULL                                /* Pointer not needed.            */
    );

    /* If window could not be created, return "failure" */
    if (!hWndMain)
        return (FALSE);

    /* Make the window visible; update its client area; and return "success"  */
    ShowWindow(hWndMain, nCmdShow);     /* Show the window                     */
    UpdateWindow(hWndMain);             /* Sends WM_PAINT message              */
    return (TRUE);                      /* Returns the value from PostQuitMessage */
}

/*****************************************************************************
 P a i n t F u n c ( )
 ============================

 This function redraws the client area of the main window.
```

→

```
 Parameter:

   HWND    Wnd:      Window handle.

 Return values:      None.

*****************************************************************************/
void PaintFunc (HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC         hDC;

    hDC = BeginPaint(hWnd, &ps); // Get display context of main window

    /* Display the message "Hello there, Windows" and the current instance */
    TextOut (hDC, 50, 50, (LPSTR)szMessage, lstrlen(szMessage));

    EndPaint(hWnd, &ps);
} // PaintFunc

/*****************************************************************************
 M a i n W n d P r o c ()
 ========================

 This function acts as the main window function. All messages are sent to
 this window.

 Parameters:

   HWND     hWnd:     Window handle.
   unsigned msg:      Message type.
   WORD     wParam:   Message-dependent 16 bit value
   LONG     lParam:   Message-dependent 32 bit value

 Return values:


   LONG                0L if the window function has sent the message.
            Otherwise, the return value from the DefWindowProc
            default window function.

*****************************************************************************/
LONG FAR PASCAL MainWndProc (HWND     hWnd,
                 unsigned msg,
                 WORD     wParam,
                 LONG     lParam)
{
   switch (msg) {
     case WM_DESTROY:
       PostQuitMessage(0);                 // Send WM_QUIT if window
       break;
```

→

โปรแกรม HELLOWIN.C (ต่อ)

```
    case WM_PAINT:                            // Client area redraw needed
      PaintFunc(hWnd);
      break;

    default: // Pass other messages to the default window function
      return (DefWindowProc(hWnd, msg, wParam, lParam));
  }
  return 0L;
}
```

■

โปรแกรม HELLOWIN.DEF

```
Model  = S
Warn   = 3

ALL: hellowin.exe

hellowin.obj: hellowin.h hellowin.c
  cl -c -A$(Model) -W$(Warn) -Gsw -Zp -Os hellowin.c

hellowin.exe: hellowin.obj hellowin.def hellowin.res
  link /NOD hellowin,hellowin.exe,NUL,libw+$(Model)libcew,hellowin.def
  rc hellowin.exe
```

โปรแกรม HELLOWIN.MAK

```
NAME          hellowin

DESCRIPTION   'Sample First Application'

EXETYPE       WINDOWS

STUB          'WINSTUB.EXE'

CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MOVEABLE MULTIPLE

HEAPSIZE      4096
STACKSIZE     4096

EXPORTS
              MainWndProc
```